L2. Problem Solving Using Search

- How to solve a problem?
- Search strategies
- Breadth-first V.S. Depth-first search
- AI programming exercise – (1)

How to solve a problem?

- Define a problem
- Known information?
- Accessible information?

➔

How to solve the problem?

What kind problem we can use search?

Define a problem

A problem is defined by four items:

initial state:

The world initial state.

operators:

A set of possible actions.

goal-test:

To check if it reaches goal state or states.

path-cost-fucntion:

It is the sum of cost of the individual actions along the path, which is from one state to another.

Notes: State space is the set of all states reachable from the initial state by any sequence of actions.

Solution is a path (a sequence of operators) from the initial state to a state that satisfies the goal test.

How to find a solution?

⇓

search

A keyword: State

- The initial state

  → (known or not? accessible or not? define-able or not?)
- The goal state

  → (known or not? accessible or not? define-able or not?)
- States transition from one state to another state

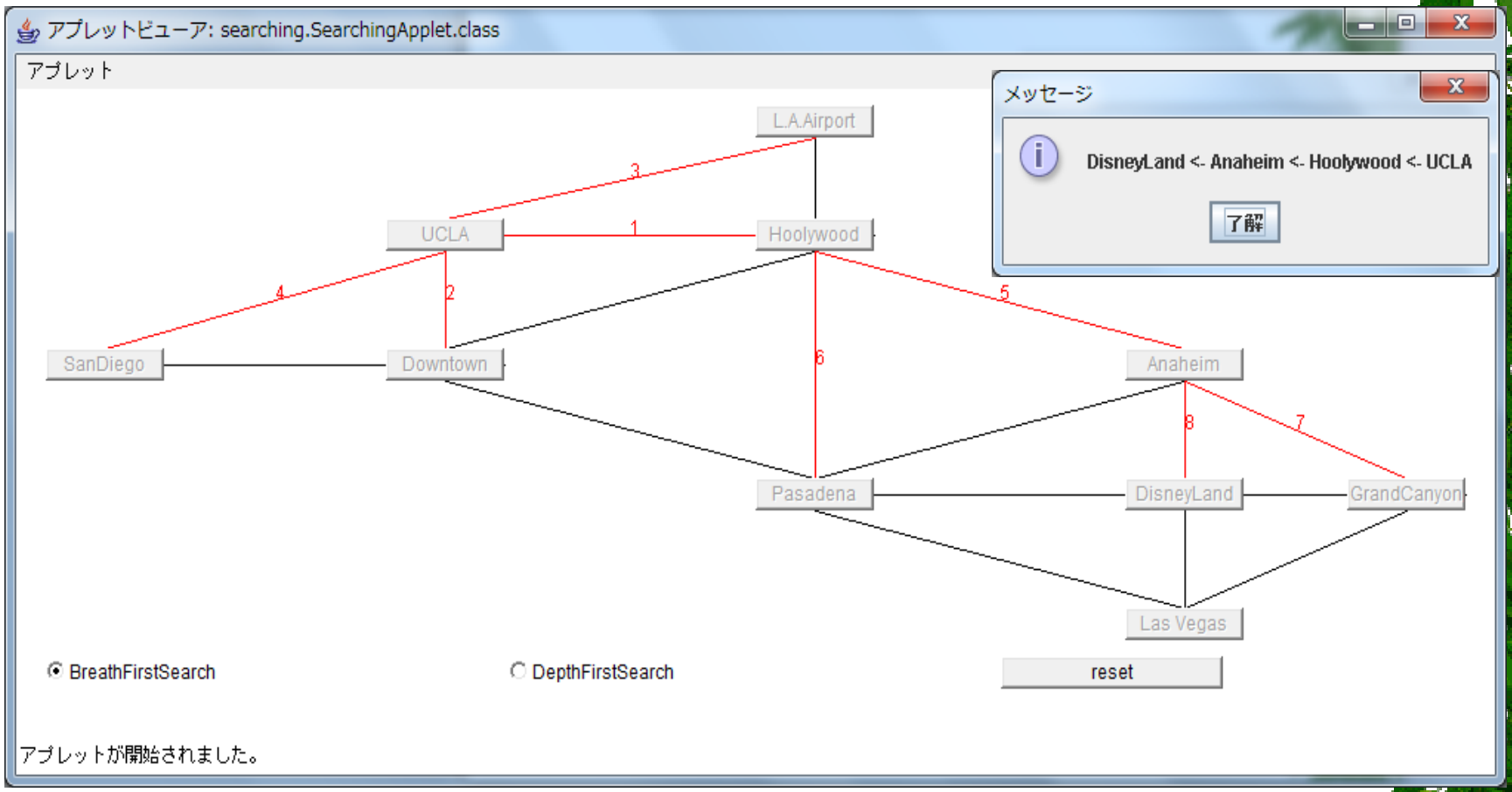
  by applying an operator

State space:

The collection of states reachable from the initial state by any sequence of actions

State space → problem types?

Some examples - e.g. 1



Initial state: UCLA

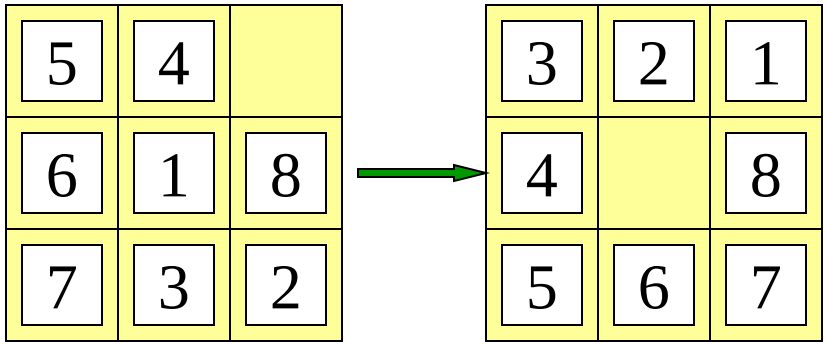
Goal state: DisneyLand

Operator: move forward

Path-cost-function: 1 in each step

Some examples - e.g. 2

The 8-puzzle



Start state Goal state

state: the location of each of the eight tiles in one of the nine squares.

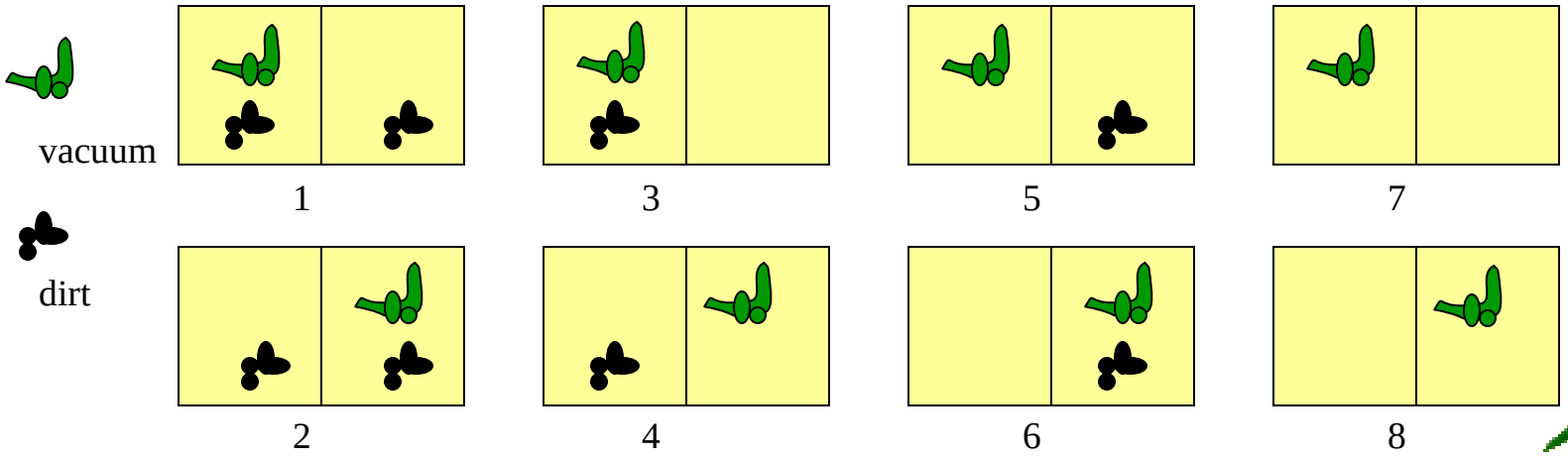
operators: blank tile moves left, right, up, or down.

goal-test: state matches the goal state.

path-cost-fucntion: each step costs 1so the total cost is the length of the path.

Some examples – e.g. 3

The vacuum world: 2 squares



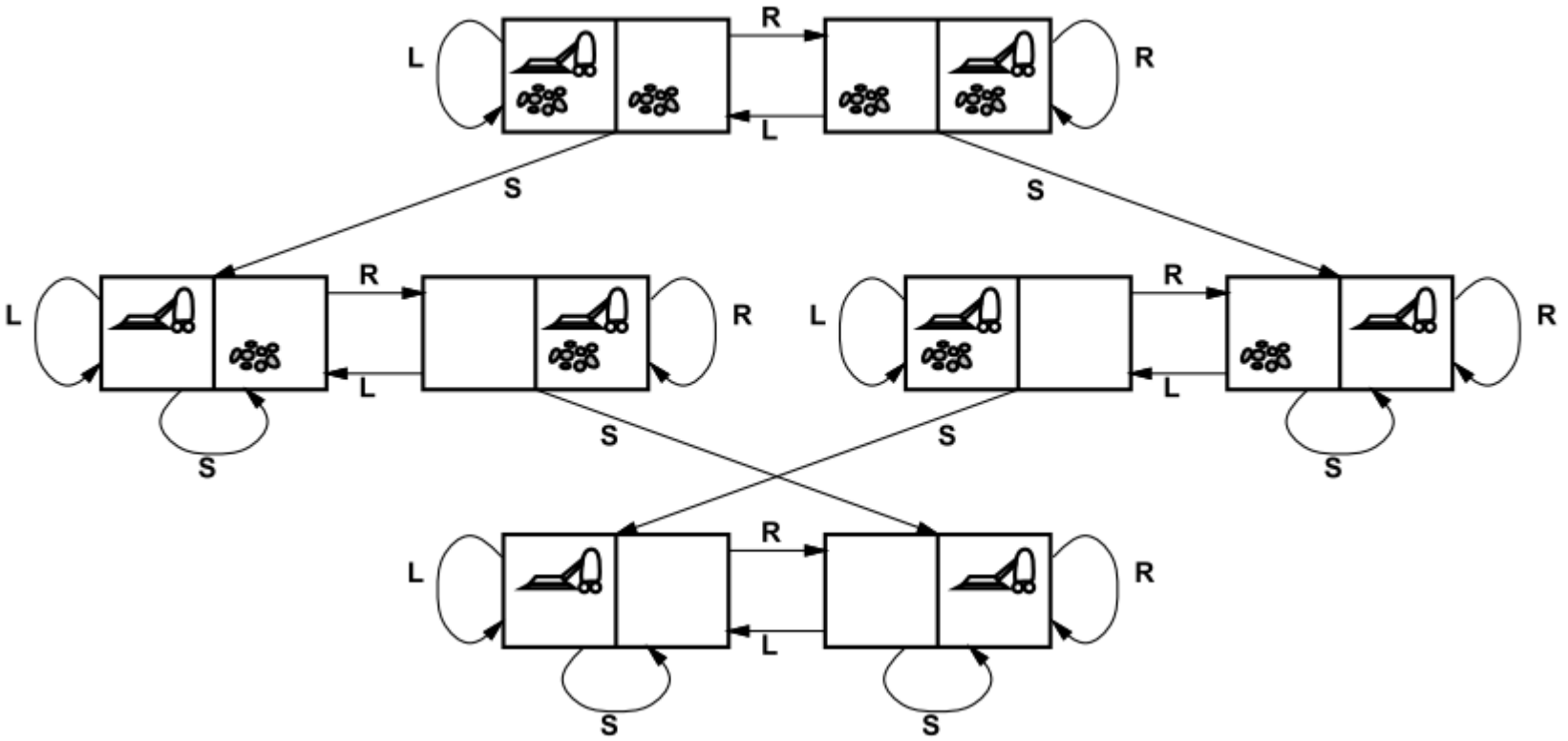
state: one of the eight states above.

operators: move left, move right, suck.

goal-test: no dirt left in any square.

path-cost-fucntion: each action costs 1.

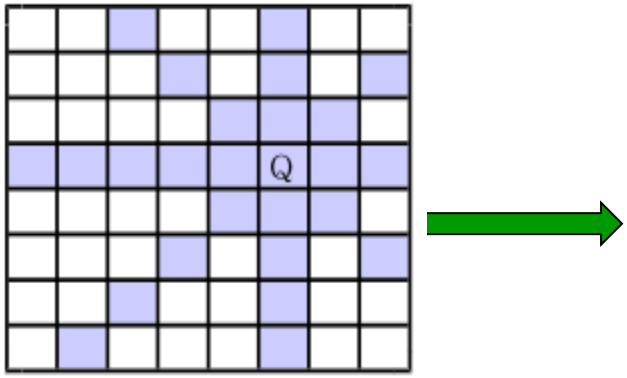
State transition graph of vacuum world

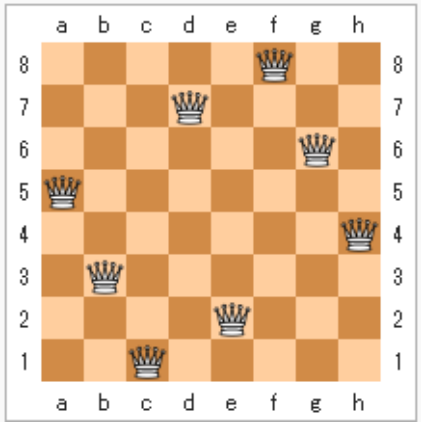


Some examples – e.g. 4

Example: *n*-queens

- Put *n* queens on an *n* × *n* board with no two queens on the same row, column, or diagonal



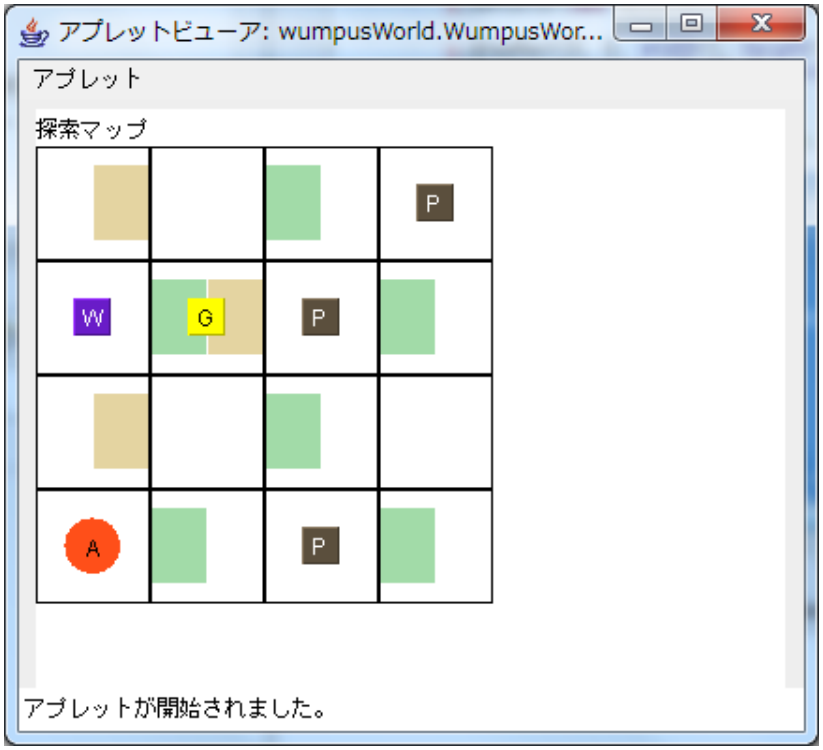


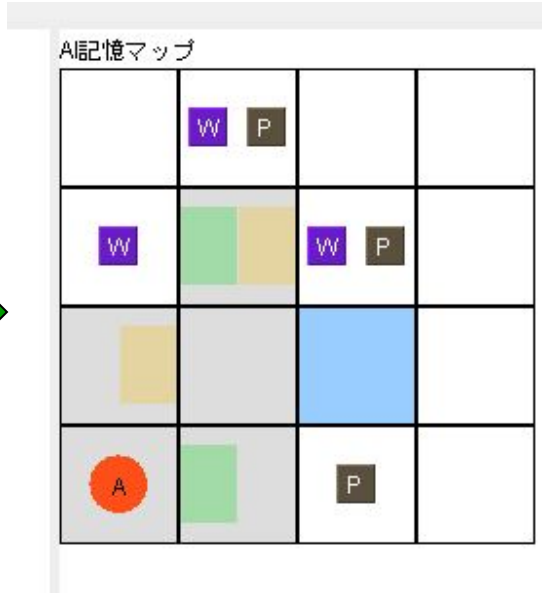
基本解は12種類ある

Work in class (write down the answers to the following questions)

- Initial state?
- Goal state?
- Operators?
- Path-cost-function?

Some examples – e.g. 5





Can you write the answers?

- Initial state -
- Goal state -
- Operators -
- Path-cost-function -

Problem types

more

knowledge

less

Deterministic, accessible => single-state problem (単一状態問題)

The agent sensors give it enough information to tell exactly which state it is in.

technique → search

Deterministic, inaccessible => multiple-state problem (多重状態問題)

The agent knows all the effects of its actions but has limited access to the world state so that it must reason about sets of states that it might get to.

technique → search + reasoning

Nondeterministic, inaccessible => Contingency problem (偶発的問題)

There is no fixed action sequence that guarantees a solution to the problem.

It must sensors during execution.

solution is a tree or policy, not a single sequence of actions.

techniques: interleave search (dynamically acquire information) + execution

Unknown state space => exploration problem (探険問題)

The agent must experiment, gradually discovering what its actions do and what sorts of states exist.

techniques: experiment + discovery

Search Algorithms

To solve the problems

- States can be defined
- The initial and goal states are known
- Operators are defined
- Path-cost-function is known

A list of search algorithms

- Breadth-first search
- Depth-first search
- Depth-limited search
- Uniform-cost search
- Iterative deepening search
- Greedy search
- Hill climbing search
- Best-first search
- ……

Breadth-first search (幅優先探索)

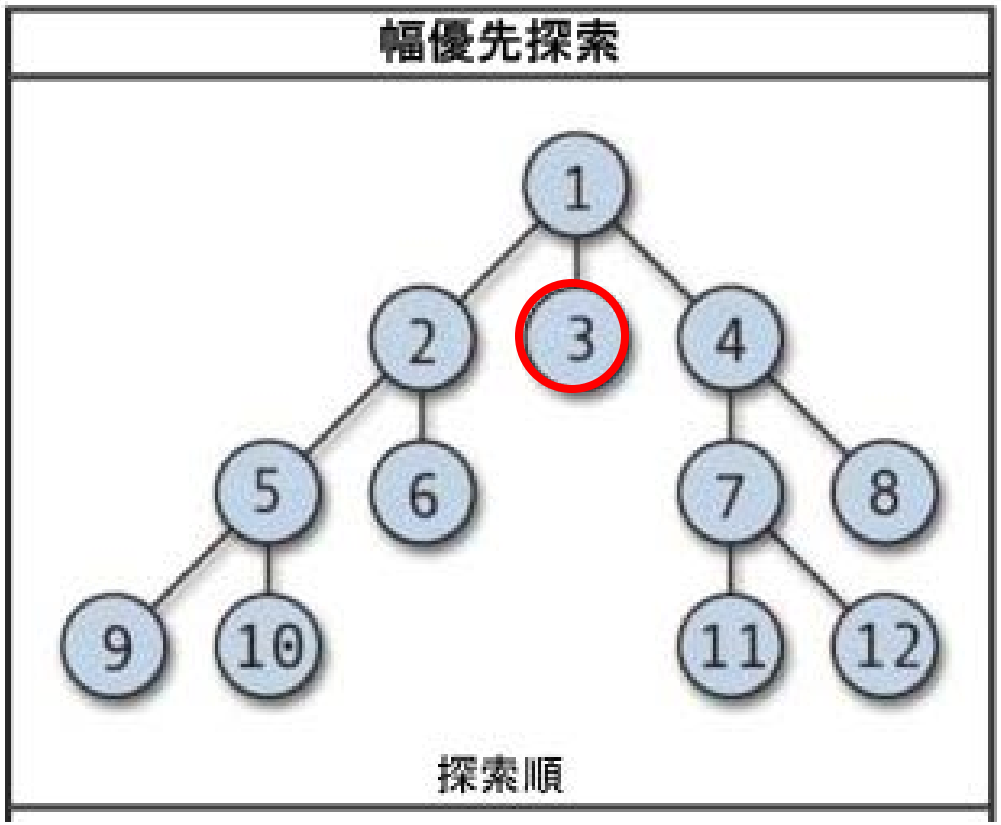
The breadth-first search algorithm searches a state space by constructing a hierarchical tree structure consisting of a set of nodes and links. The algorithm defines a way to move through the tree structure, examining the value at nodes. From the start, it looks at each node one edge away. Then it moves out from those nodes to all two edges away from the start. This continues until either the goal node is found or the entire tree is searched.

幅優先探索アルゴリズムはノードやリンクからなる階層的なツリー構造を構成することにより、状態空間を検索します。アルゴリズムはツリー構造を通して移動すると定義されています。スタートからそれぞれのノードを見ます。それからスタートから繋がっている二つのノードへと移動する。ゴールノードが見つかるか、すべてのツリーが探索されるまでこの操作を続けます。
※ある階層をすべて調べ、それが終わるとひとつ深い階層をすべて調べることを繰り返す。

The algorithm follows:

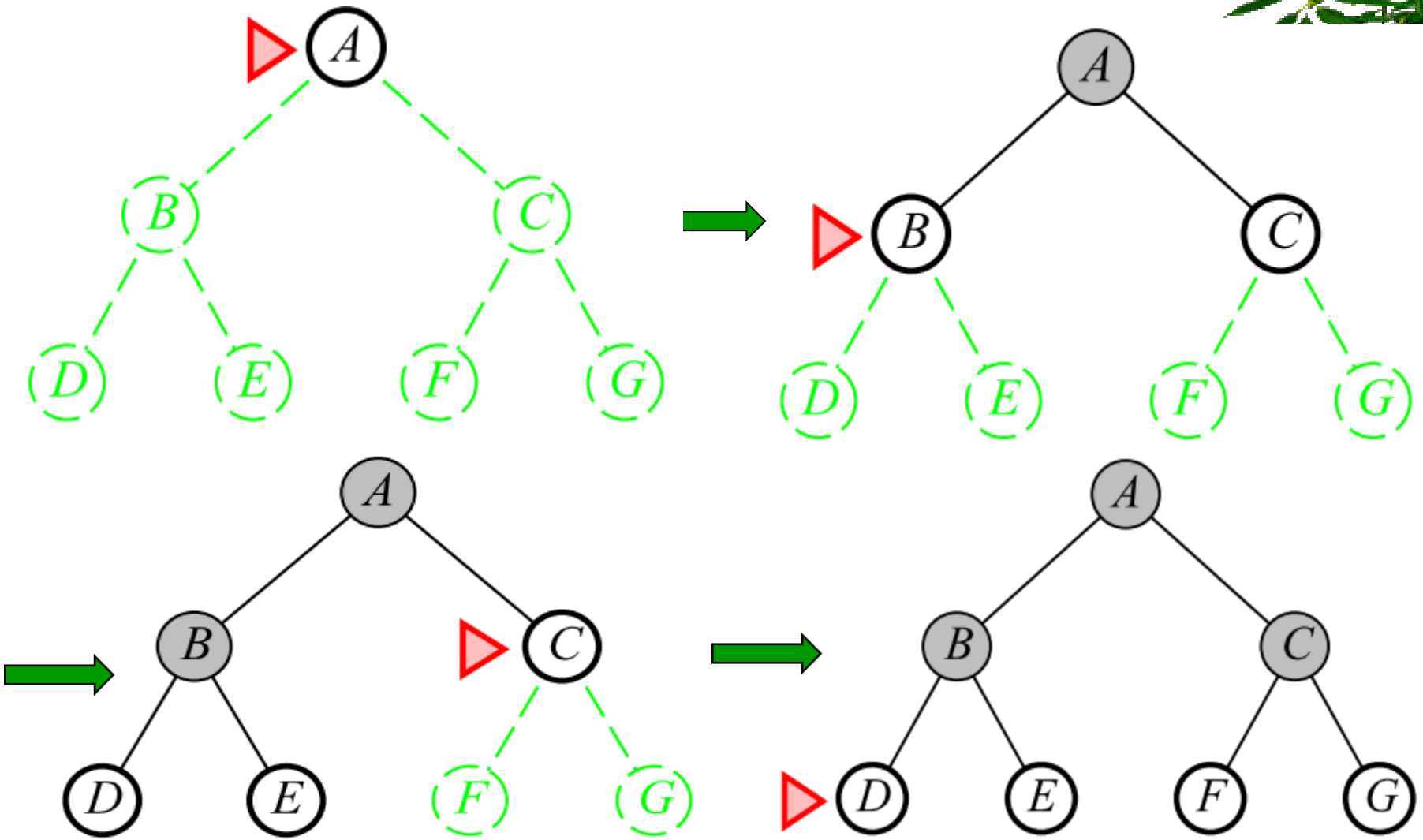
1. Create a queue and add the first **node** to it.
2. Loop:
   - If the queue is empty, quit.
   - Remove the first **node** from the queue.
   - If the **node** contains the goal state, then exit with the node as the solution.
   - For each child of the current node: add the new state to the **back** of the queue.

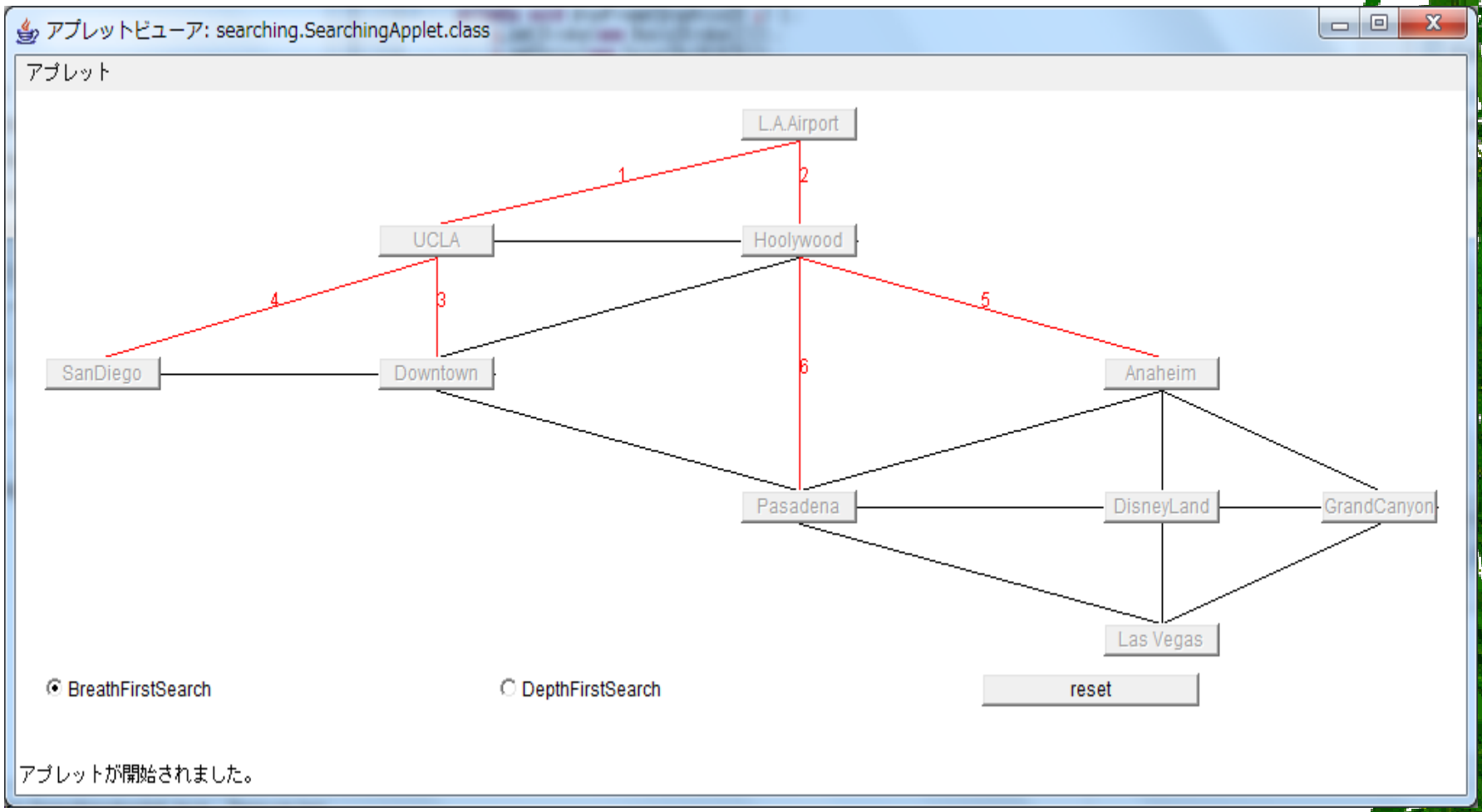
An example:

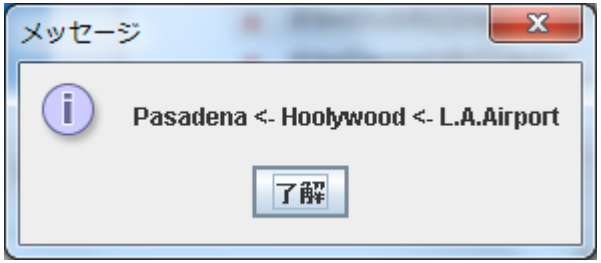


1-> 2->3->4->5->6 -> ……

FIFO queue is used in the implementation, children nodes are added at end.




アプレットビューア: searching.SearchingApplet.class
アプレット
L.A.Airport
1
2
UCLA
Hoolywood
4
3
5
SanDiego
Downtown
6
Anaheim
Pasadena
DisneyLand
GrandCanyon
Las Vegas
BreathFirstSearch
DepthFirstSearch
reset
アプレットが開始されました。


メッセージ
Pasadena <- Hoolywood <- L.A.Airport
了解

Depth-first search(深さ優先探索)

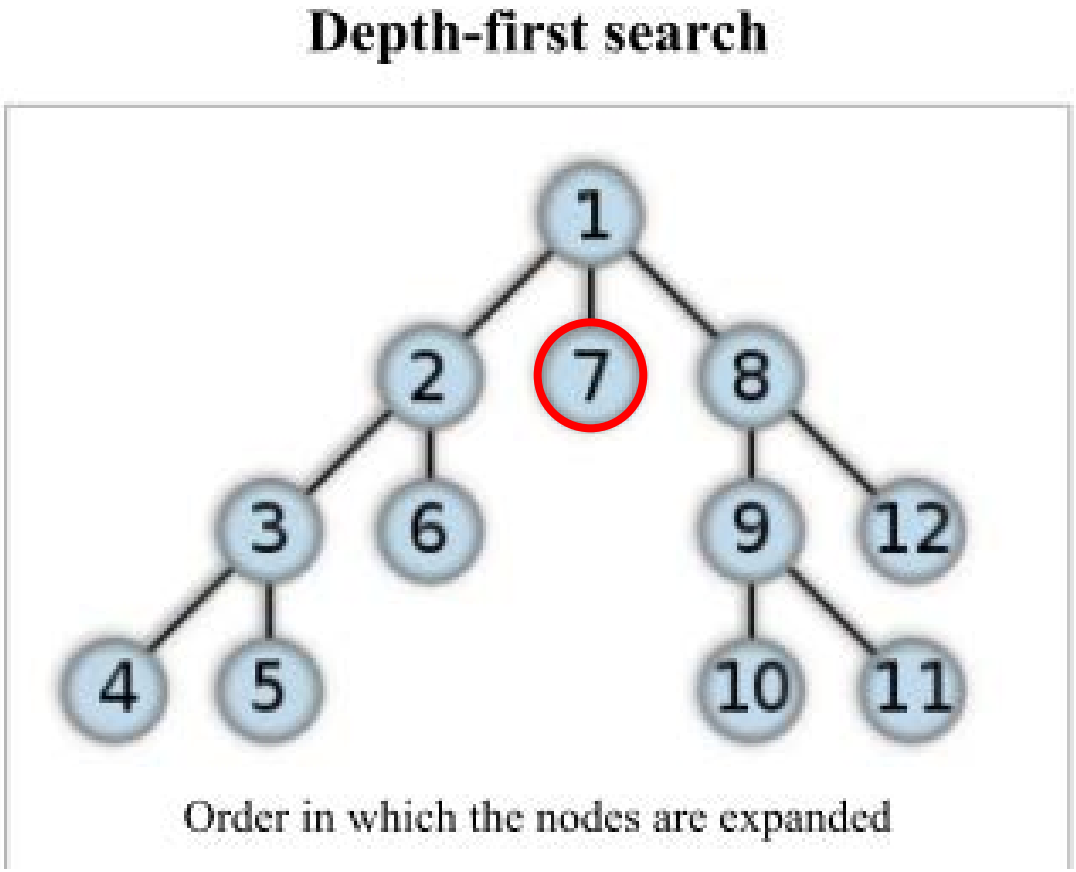
The depth-first algorithm follows a single branch of the tree down as many levels as possible until we either reach a solution or a dead end. It searches from the start or root node all the way down to a leaf node. If it does not find the goal node, it backtracks up the tree and searches down the next untested path until it reaches the next leaf.

深さ優先探索アルゴリズムは木の最初のノードから、目的のノードが見つかるか行き止まりまで深く下がっていきます
スタートか根ノードから始まり、ずっと下ったところの葉ノードまで探索する。もしゴールノードが見つからなかったら引き返し、次のまだ通っていないツリーを葉に到達するまで探す。
Root node ： ツリー構造の頂点にあるノード
leaf node ： ツリー構造の最底辺にあるノード
※ 深く進んでいき、行き止まったら引き返して、また深く進みながらゴール目指す

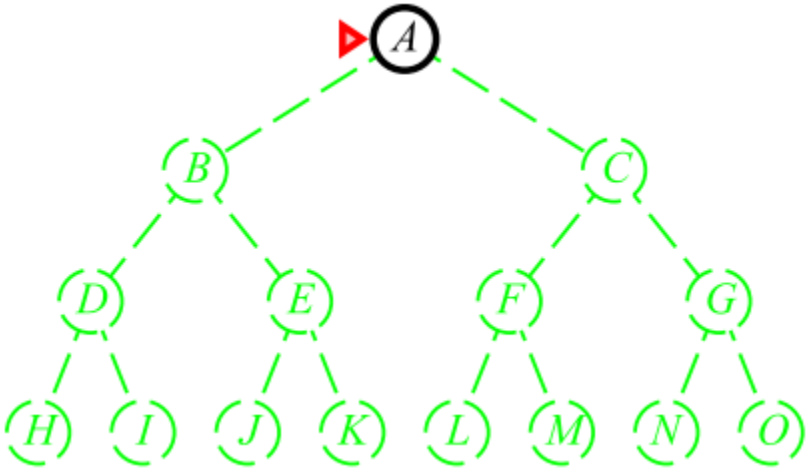
The algorithm follows:

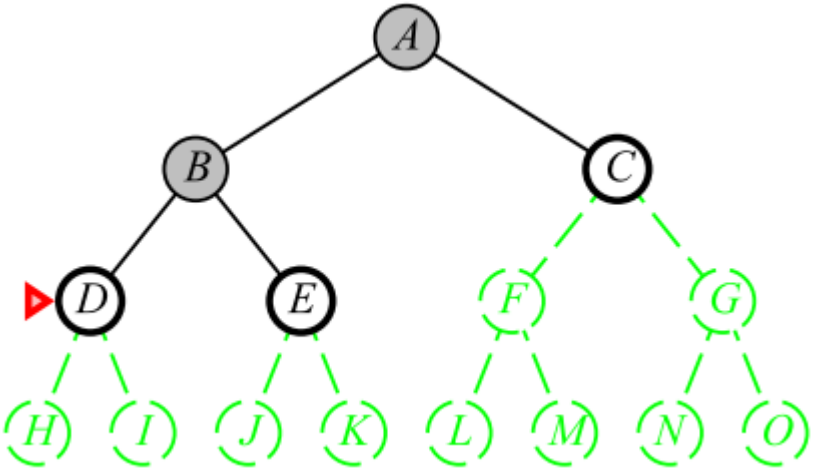
1. Create a queue and add the first **node** to it.
2. Loop:
   - If the queue is empty, quit.
   - Remove the first **node** from the queue.
   - If the **node** contains the goal state, then exit with the node as the solution.
   - For each child of the current node: add the new state to the **front** of the queue.

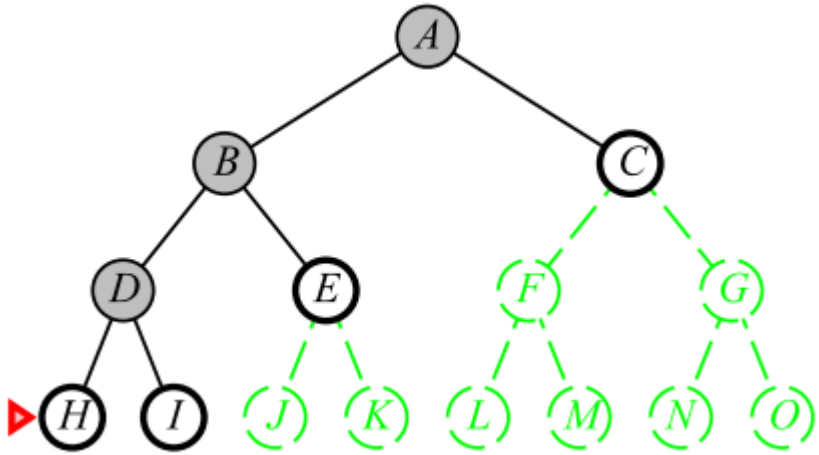
An example:

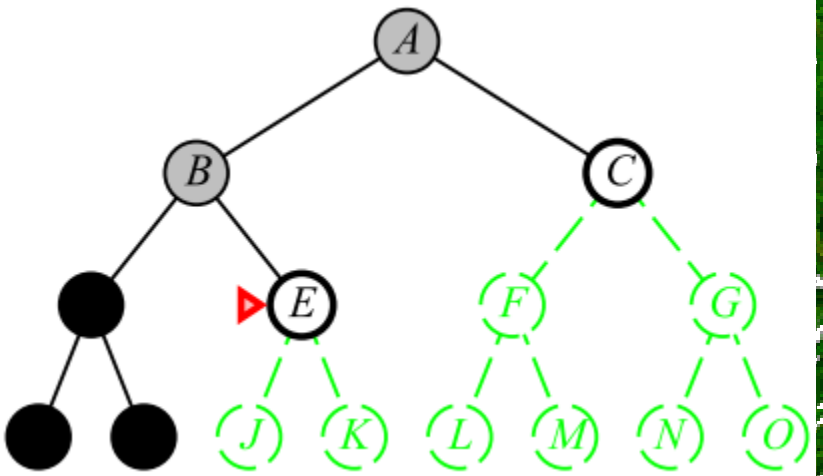

Depth-first search
1
2
7
8
3
6
9
12
4
5
10
11
Order in which the nodes are expanded

LIFO queue is used in the implementation, children nodes are added at front.









Define a Node class → Construct a tree or a graph → Searching

1 2 3

1

```
class Node {
    String name;
    Vector children;
    Node pointer; // 解表示のためのポインタ

    Node(String theName){
            name = theName;
            children = new Vector();
    }

    ……
}
```

Vector → ArrayList<Node>

```
import java.util.*;
public class Search{
    Node node[];
    Node goal;
    Node start;
    Search(){
        makeStateSpace();
    }
    private void makeStateSpace(){
        …
    }
    public void breadthFirst(){
        …
    }
    public void depthFirst(){
        …
    }
    public void printSolution(Node theNode){
        …
    }
    public static void main(String args[]){
        …
        (new Search()).breadthFirst();
        (new Search()).depthFirst();
        …
    }
```

2

3

3

Vector → ArrayList<Node>

1

```
class Node {
    String name;
    Vector children;
    Node pointer; // 解表示のためのポインタ
    Node(String theName){
                name = theName;
                children = new Vector();
    }
    public String getName(){
                return name;
    }
    public void setPointer(Node theNode){
                this.pointer = theNode;
    }
    public Node getPointer(){
                return this.pointer;
    }
   public void addChild(Node theChild){
                children.addElement(theChild);
    }
    public Vector getChildren(){
                return children;
    }
    public String toString(){
                String result = name;
                return result;
    }
}
```

2

```
private void makeStateSpace(){
    node = new Node[10];

    node[0] = new Node("L.A.Airport");
    start = node[0];
    node[1] = new Node("UCLA");
    node[2] = new Node("Hoolywood");
    node[3] = new Node("Anaheim");
    node[4] = new Node("GrandCanyon");
    node[5] = new Node("SanDiego");
    node[6] = new Node("Downtown");
    node[7] = new Node("Pasadena");
    node[8] = new Node("DisneyLand");
    node[9] = new Node("Las Vegas");
    goal = node[9];

    node[0].addChild(node[1]);
    node[0].addChild(node[2]);
    node[1].addChild(node[2]);
    node[1].addChild(node[6]);
    node[2].addChild(node[3]);
    node[2].addChild(node[6]);
    node[2].addChild(node[7]);
    node[3].addChild(node[4]);
    node[3].addChild(node[7]);
    node[3].addChild(node[8]);
    node[4].addChild(node[8]);
    node[4].addChild(node[9]);
    node[5].addChild(node[1]);
    node[6].addChild(node[5]);
    node[6].addChild(node[7]);
    node[7].addChild(node[8]);
    node[7].addChild(node[9]);
    node[8].addChild(node[9]);
}
```

```
public void breadthFirst(){
   Vector open = new Vector();
   open.addElement(node[0]);
   Vector closed = new Vector();
   boolean success = false;
   int step = 0;
   for(;;){
      System.out.println("STEP:"+(step++));
      System.out.println("OPEN:"+open.toString());
      System.out.println("closed:"+closed.toString());
      if(open.size() == 0){// openは空か？
         success = false;
         break;
       } else {
         Node node = (Node)open.elementAt(0);
         open.removeElementAt(0);
         if(node == goal){
            success = true;
            break;
         } else {
            Vector children = node.getChildren();
            closed.addElement(node);
            for(int i = 0 ; i < children.size() ; i++){
               Node m = (Node)children.elementAt(i);
               if(!open.contains(m) && !closed.contains(m)){
                  m.setPointer(node);
                  if(m == goal){
                     open.insertElementAt(m,0);
                  } else {
                     open.addElement(m); //add at the ack
                    } } } } } }
    if(success){
       System.out.println("*** Solution ***");
       printSolution(goal);
    }
}
```

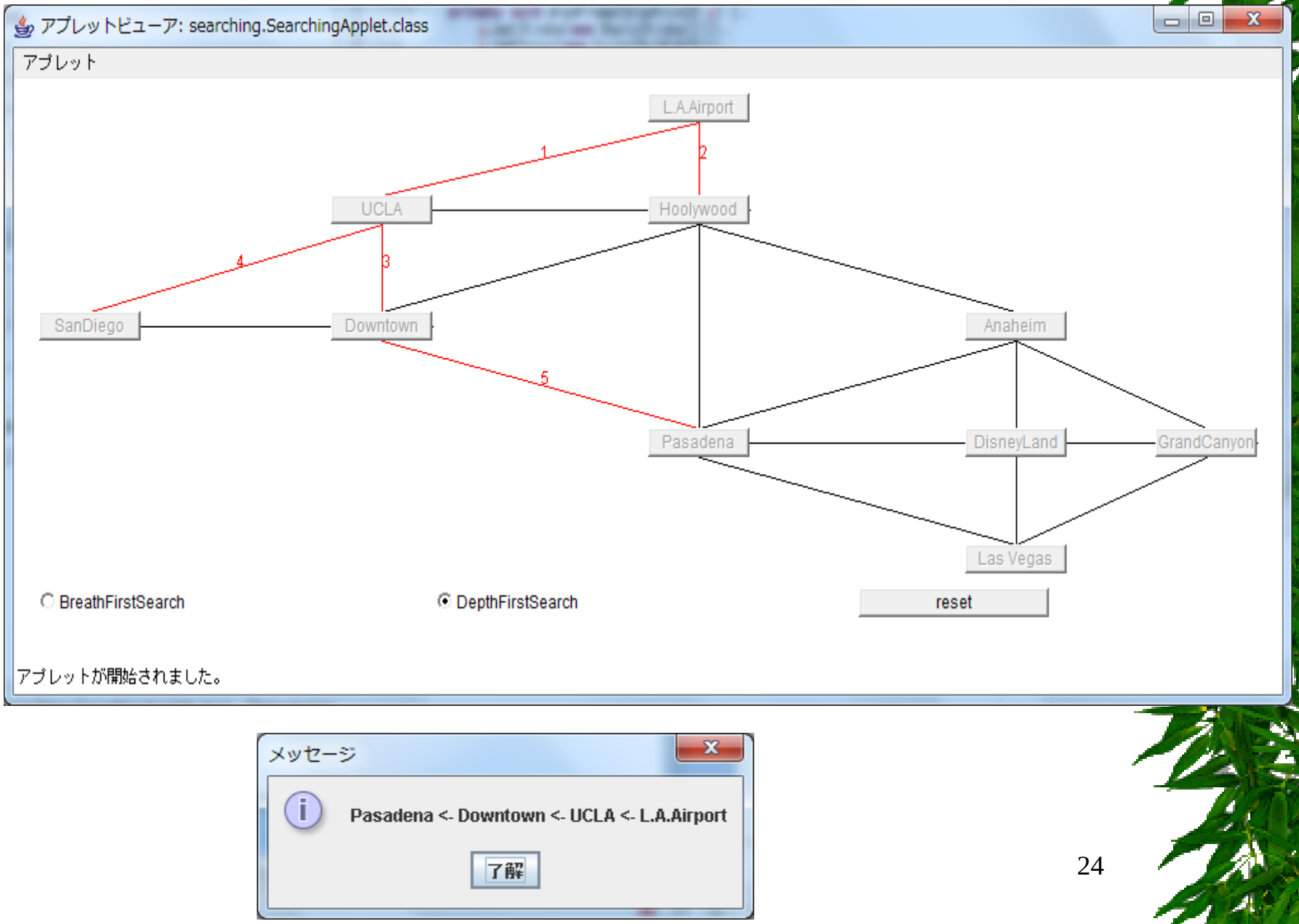
Vector → ArrayList<Node>

3

```
public void depthFirst() {


// Add your code here


}
```


アプレットビューア: searching.SearchingApplet.class
アプレット
L.A.Airport
1
2
UCLA
Hoolywood
4
3
SanDiego
Downtown
Anaheim
5
Pasadena
DisneyLand
GrandCanyon
Las Vegas
BreathFirstSearch
DepthFirstSearch
reset
アプレットが開始されました。
メッセージ
Pasadena <- Downtown <- UCLA <- L.A.Airport
了解

Work in class

Take a look this sites

http://codereview.stackexchange.com/questions/48518/depth-first-search-breadth-first-search-implementation

Find your favor reference.

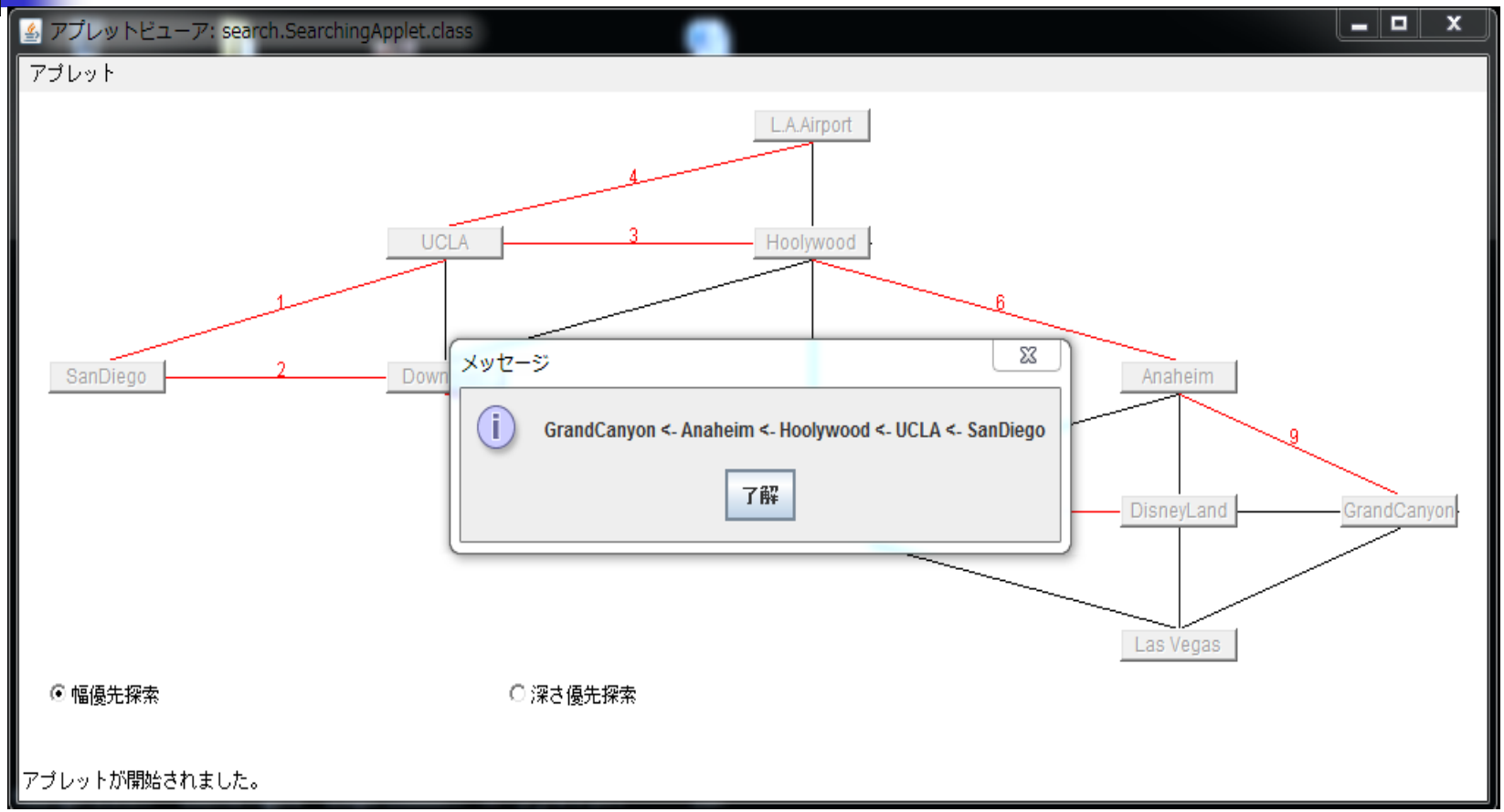
Think about how to make your own search program. .

探索とは

- 探索とは人工知能の分野における基本的な要素技術であり、今ではカーナビやゲームのAIなど、幅広い分野で応用されている
- おおまかに言うと、ある入力に対しいくつもの解を評価して結果を返すこと
- この講義では、一般的な幅優先探索と深さ優先探索について学ぶ

AI 応用の例 – 探索問題

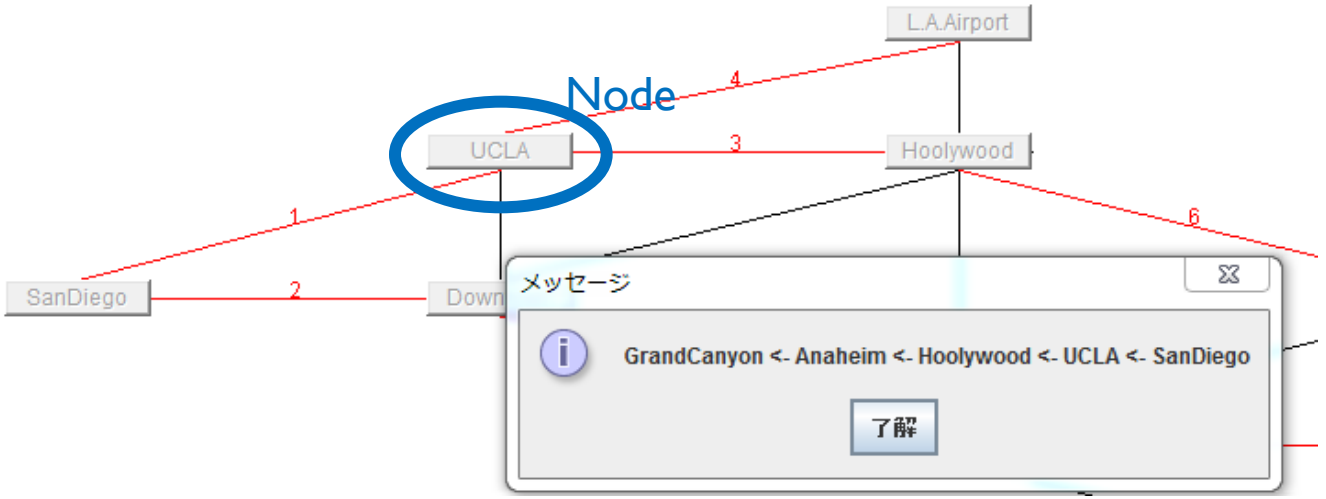
AI 応用の例 – 探索問題


アプレットビューア: search.SearchingApplet.class
アプレット
L.A.Airport
4
UCLA
3
Hoolywood
1
6
SanDiego
2
Down
Anaheim
9
DisneyLand
GrandCanyon
Las Vegas
メッセージ
GrandCanyon <- Anaheim <- Hoolywood <- UCLA <- SanDiego
了解
幅優先探索
深さ優先探索
アプレットが開始されました。

Nodeの作成と表示

- Nodeに必要なものを考えてみましょう

（街の）名前
ボタン
X座標
Y座標
経路情報
隣接する町情報
隣接する町に引く線



Node.javaクラスの実装例

```
import java.awt.Button;
import java.util.ArrayList;
import java.util.HashMap;

//都市クラス
public class Node extends Button {
// 町の名前
String name;
// 座標
private int x, y;
// 幅と高さ
private int w = 80, h = 20;
// 隣接都市
ArrayList<Node> children;
// 経路
Node pointer;
// 隣接都市への線(隣接都市/隣接都市につなぐ線)
HashMap<Node, Line> hm;
```

Nodeの表示，隣接するNode

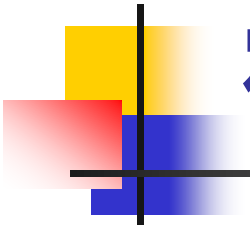
- ShowNode.javaを書き加えていくつかのNodeをApplet上に表示し、Nodeをクリックすると都市名をダイアログで表示するようにしてみましょう
- Line.java

  隣接するNodeに引く線

探索とは

- 探索とは人工知能の分野における基本的な要素技術であり、今ではカーナビやゲームのAIなど、幅広い分野で応用されている
- おおまかに言うと、ある入力に対しいくつもの解を評価して結果を返すこと
- この講義では、一般的な幅優先探索と深さ優先探索について学ぶ

AI 応用の例 – 探索問題



宿題

Make your own search program

(1) Run in console

(2) Run in GUI